Paper

On the Influence of Network Impairments on YouTube Video Streaming

Arkadiusz Biernacki[a], Florian Metzger[b], and Kurt Tutschku[b]

[a] *Institute of Computer Science, Silesian University of Technology, Gliwice, Poland*
[b] *Chair of Future Communication, University of Vienna, Vienna, Austria*

Abstract—Video sharing services like YouTube have become very popular which consequently results in a drastic shift of the Internet traffic statistic. When transmitting video content over packet based networks, stringent quality of service (QoS) constraints must be met in order to provide the comparable level of quality to a traditional broadcast television. However, the packet transmission is influenced by delays and losses of data packets which can have devastating influence on the perceived quality of the video. Therefore, we conducted an experimental evaluation of HTTP based video transmission focusing on how they react to packet delay and loss. Through this analysis we investigated how long video playback is stalled and how often re-buffering events take place. Our analysis revealed threshold levels for the packet delay, packet losses and network throughput which should not be exceeded in order to preserve smooth video transmission.

Keywords—multimedia communication, network measurements, quality of service, video streaming.

1. Introduction

During the past years video sharing services like YouTube in the US, Smiley in Japan, the now defunct Megavideo in Hong-Kong, and Dailymotion in France have become very popular. YouTube users alone request millions of videos every day. Consequently, popularity of this kind results in a drastic shift in Internet traffic statistic, which reports that the share of P2P traffic is declining, primarily due to an increase in traffic from Web-based video sharing services [1]. So far, there is no indication that this trend will decrease and indeed is more likely to sustain. Thus, fulfilling the rising demand for video traffic will be a challenging task for both content providers as well as ISPs (Internet Service Providers).

Video streaming in the above mentioned services is either web-based or HTTP-based, therefore being transported using the TCP. The TCP is currently the most widely used transport protocol in the Internet but conventionally regarded as inappropriate for media streaming. The primary reason lies in the TCP reliability and retransmission mechanisms which can lead to undesirable transmission delays and may violate timeliness requirements for streamed live media. In this context, coping with packet delay and loss, which can occur due to congestion or packet corruption, demands new solutions as classical transmission procedures, used in unreliable protocols, e.g. UDP, may be not sufficient. It should also be taken into account that the HTTP and TCP are general purpose protocols and were not specifically designed or optimized for streaming media delivery. Thus, attempts are being made to adapt media delivery to the Internet instead of trying to adapt the Internet to multimedia content streaming.

In our work we concentrate on YouTube which represents a service that is unlike the traditional VoD systems in several important aspects. From our perspective, the most important difference between YouTube and other more traditional VoD systems is that the latter usually offer professionally-produced video content such as movies, news, sport events, or TV series. The quality and popularity of this content are well-controlled and predictable. In contrast, YouTube videos can be uploaded by anyone with access to the Internet. The quality of these video clips vary significantly making network optimizations for specific content unreasonable.

Internet connections are characterized by a number of statistically determined characteristics including latency and reliability. These traits are not guaranteed – in fact, they can fluctuate considerably depending on the local ISP network load, remote server load, background traffic, as well as network infrastructure quality. Video delivered by more traditional channels such as satellite, DVD, cable or digital TV broadcasting requires usually not to much buffering space at a client side because data arrives at a media player with mostly deterministic delay, rate and very limited or infrequent data drops. Video delivered over the Internet is much more problematic because there is no guarantee that the data will flow to a user at a sufficient rate and determined delay. Instead, it arrives with a rate and delay that can change consistently during video file transmission. Therefore, buffering is of increasing importance for video streams when they are transmitted over the Internet, including Web-based streaming. The YouTube client software manages buffering and playing of the received content using several behaviors [2].

In our work we study the efficiency of three possible streaming client playback strategies. Our goal is to investigate how often the player buffer runs out under different network interferences, especially packet loss and delay. As a consequence, we want to investigate how these parameters influence the perceived quality of video received by end users. An ISP may have partial influence on these characteristics and therefore may be able to tune the quality of video transfer and influence its users' satisfaction.

2. Video Distribution

2.1. Protocols

Contemporary media delivery systems can be classified into two categories: systems with and systems without feedback control mechanism.

One of the options for multimedia delivery systems with feedback control is an usage of the Real-Time Streaming Protocol (RTSP). The RTSP is a stateful protocol, which means that the server keeps track of the client's state from the first time the client connects to the streaming server until the time it disconnects. The client communicates its state to the server by sending commands such as play, pause or disconnect. The server begins sending the media as a steady stream of small RTP (Real-time Transport Protocol) packets. The data is sent at the media bitrate and the client buffer is filled with just few packets before playback begins. If the client or server discover any interferences in their communication, like increasing latency or packet drops, they can renegotiate transmission parameters, e.g., the server can send the same video content but with reduced encoding rate. The transmission is usually based on unreliable transport protocols, most commonly the UDP. However, when using the UDP, data packets often have difficulty getting around firewalls and network address translators. Thus, sometimes the TCP is preferred when firewalls or proxies block UDP packets, although at the expense of potentially unnecessary reliability.

Such problems are limited when using the HTTP as a media delivery protocol because firewalls and routers know how to pass HTTP traffic through. It also does not require special proxies or caches. The HTTP is a stateless protocol. Thus, multimedia transmission based on it share this feature and behave as a system without feedback control. Basically, if an HTTP client requests data, the server responds by sending the required data, but it does not remember the client or its state which means that each HTTP request is handled completely independently.

HTTP streaming may be implemented in several ways. In our work we focus on an implementation which can be described as a progressive download. The progressive download is nothing more than a transfer of a video file from a HTTP server to a client where the client may begin playback of the file before the download is complete. Contrary to the above mentioned systems with feedback control, which rarely send more than a few seconds of video content to a client in advance, HTTP streaming (web) servers progressively push the whole video content to a client, usually does not taking into account how much data have been already sent in advance. Simultaneously, most players are capable of playing the video file while its download is still in progress. Most web-based streaming platforms, including Vimeo, MySpace, and MSN Soapbox, are based on HTTP and do not have a feedback control. However, some HTTP streaming services, e.g. YouTube, implement additional application layer flow control mechanisms that limit the transmission rate to the same magnitude as the video bitrate [3].

Currently, it is thought that HTTP media streaming is easier and cheaper to deploy because web streaming can use generic HTTP solutions and does not require specialized servers at each network node. Standard HTTP caching mechanism allow to move media content to an edge of the network, closer to users. Nonetheless, the above technology have also shortcomings. The congestion avoidance algorithm of the TCP produces a saw-tooth shaped transmission rate. Furthermore, the reliability of TCP results in variable transmission delays due to retransmissions of lost packets. As a consequence, it was commonly assumed that the TCP is not suitable for multimedia streaming, which is to some extent loss tolerant but delay sensitive. The instantaneous transmission rate and transmission delay variation of the TCP must be smoothed out by receiver-side buffering. Despite these drawbacks, currently a dominant share of multimedia traffic is being delivered using the HTTP and TCP [1].

2.2. Video Buffering

Most of HTTP players are able to concurrently play and download the same file. In the simplest case the player fills its internal buffer at the beginning of the video transmission and starts the video playback as soon as a minimum buffer level is achieved. While simultaneously playing and downloading the content, the amount of video data in the buffer is variable and depends mainly on the download bandwidth, video bitrate and video playing rate. When the download bandwidth is larger than the video rate the buffer grows. In the opposite case, the buffer will shrink and if the situation last long enough it may also run out. In such cases the video stalls and the player waits until the buffer will be refilled again.

Let's assume that $G(t)$ represents the number of data packets generated at a HTTP server by time t, Fig. 1 with a packet transmission rate limited only by the infrastructure conditions like TCP throughput, server performance, etc. The first packet is generated at time 0 and sent immediately to a client. Let $A(t)$ denote the number of packets arriving at the client by time t and $B(t)$ denote the number of packets played by the client by time t. Since the transmis-

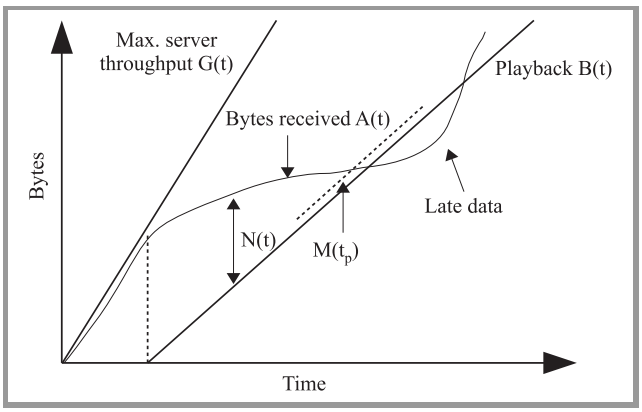


Fig. 1. Player buffer occupancy in the time function.

sion rate is constrained by the generation rate at the server, we have $A(t) \leq G(t)$. A packet arriving earlier than its playback time is referred to as an early packet. At time t, the number of early packets is counted as $N(t) = A(t) - B(t)$. A negative value of $N(t)$ indicates that the packet arrival is behind the playback by $-N(t)$ packets.
During streaming, there can be many time periods Δt_i for which $N(\Delta t_i)$ has a negative value. In our work we try to answer the question: what is the value of $\sum_i \Delta t_i$ and i, i.e. the total video stall time in a relation to video clip length and the number of stalling events for several video files transmitted from YouTube.

2.3. Video Playing Strategies

For video streaming YouTube currently uses amongst others device-dependent 3GP containers with RTSP dedicated for mobile streaming applications and Adobe Flash with HTML5 employing HTTP streaming of Flash Video. Adobe Flash, henceforth referred to as Flash, is the default container when YouTube is accessed via a PC. Users need to install a proprietary plug-in for viewing Flash videos. HTML5 supports videos that do not require any proprietary plug-ins running directly.
When streaming with the Flash Player, it basically behaves like a simple HTTP player described above i.e. it starts the video playback as soon as a minimum buffer level is achieved. However, thanks to the flexibility of the Flash authoring platform, the buffering functionality can be additionally enhanced using client-side ActionScript code. The standard buffering process is believed to be susceptible to bandwidth drops, as well as being unable to exploit a sudden increase of bandwidth. The enhancement is called a dual-threshold buffering strategy and assures a faster start and, at the same time, should provide better resilience to bandwidth fluctuations, or other adverse network conditions. Therefore, the playback of a video file starts when the first threshold in the buffer is filled with a given amount of data. But, instead of trying to keep the buffer full to this level, the modified strategy attempts to fill the buffer to a second, higher threshold. This additional data may be useful later if the network connection encounters temporary impairments like bandwidth drops or fluctuations.
In the case of HTML5 streaming, the playing strategy depends on particular video player implementation. The W3C HTML5 specification [4, Section 4.8] states, that in the case of autoplay "the user agent [...] will automatically begin playback of the media resource as soon as it can do so without stopping". To approximate this difficult to fulfil condition every implementation differs. We investigated Firefox's implementation of this spec as its code is open source and the behaviour can therefore be studied not just by observing network traces but also by reading the sources. The algorithm in the Firefox is summarized in algorithm in Fig. 2 and Table 1. Rather than using static thresholds it facilitates moving averages to estimate the development of the transmission rate. It does not differentiate between the initial video startup time and intermittent buffering events.

if $s_{MA} > v_{MA}$ **then**
    $c \leftarrow (b_b = 20s \vee b_T = 20s)$
else
    $c \leftarrow (b_b = 30s \vee b_T = 30s)$
end if

Fig. 2. Firefox playback (re-)start decision algorithm.

This implementation requires large playback buffers due to the chosen high video buffering amounts, but could also result in very few stalling events.

Table 1
Variables involved in buffering decisions

Variable	Explanation
s_{MA}	Moving average of the transmission speed.
v_{MA}	Moving average of the video bitrate.
c	Condition upon which to start/resume playback.
b_b	Amount of video data the buffer contains.
b_T	Amount of time spent in non-playing buffering state.

The HTML5 network traffic also differs from the Flash traffic. The works [5] and [2] identified YouTube's block transmission behaviour, which uses longer and client application controlled block phases for Google Chrome and no blocking at all for Firefox.

3. Previous Works

A major research area related to our work is concerned with the analysis and characterization of streaming services in the Internet. Early works in this area go back to the twentieth century and focused amongst others on the characterization of videos on the Web [6], video access statistics of users [7], developing UDP-based streaming protocols and providing mechanisms for TCP-friendliness and loss recovery, e.g. [8], [9].
When to concentrate on the HTTP video, several YouTube measurement studies have been reported in literature in the last few years. These works focused on characterizing various aspects of YouTube videos, as well as its usage patterns. On the one hand, we have work based on user traffic trace analysis including deep packet inspection, e.g. [2], [10]–[12]. Their authors operated on real world measurements obtained from, e.g., ISPs' networks and they characterized video popularity, durations, size and playback bitrate, as well as usage pattern statistics such as day versus night patterns or traffic volume. Additionally, in [10] the investigation of YouTube user sessions statistics was presented. In [2] the authors presented a traffic characterization of Netflix and YouTube, and identified different streaming strategies deriving also a model for the aggregate traffic generated by these services. Plissonneau *et al.*

in [12] described the impact of YouTube traffic on a French regional ADSL point of presence revealing that YouTube video transfers are faster and larger than other large Web transfers.

On the other hand, there are publications based on crawling the YouTube site for an extended period of time [13]–[15]. These works examined video popularity and user behaviour and found that statistics such as length, access patterns, growth trend, and active life span were quite different compared to traditional video streaming applications. Furthermore, in [13] information directly available from YouTube servers was used to analyse the characteristics of videos served by YouTube while [14] investigated social networking in YouTube videos. Also Abhari and Soraya in [15] investigated YouTube popularity distribution and access patterns through the analysis of a vast amount of data collected by crawling the YouTube API. On the basis of the observations, the authors presented essential elements of the workload generator that can be used for benchmarking caching mechanisms.

A global study of user experience for YouTube videos using PlanetLab nodes from all over the world is performed in [16]. Results from this analysis show that on average there are about 2.5 pauses per video, and on average 25% of the videos with pauses have total pause time greater than 15 seconds.

The closest work to ours is [17] where the authors evaluated the responsiveness of adaptive HTTP algorithms (taking into account YouTube amongst others) under variable network conditions. The authors claimed that the performance of the streaming algorithm increases with the decrease of network delay and by providing information to the client, particularly about the achievable throughput. It compensates for the structural noisiness of measurements and improves the ability of the client to accurately estimate the throughput.

4. Experiments

In order to observe the behaviour of YouTube, and, for that matter, any other, streaming under varying network conditions, we created a controlled, isolated environment for our tests, depicted in Fig. 3. Analyses are conducted in two phases. In phase one, Python scripts on a client computer simulate a streaming application by issuing HTTP GET requests to videos at a YouTube cache. Any video data is stored and analysed for its frame characteristics, such as the size, type and relative playback time. Furthermore, the client captures the packet trace using tcpdump/libpcap, allowing offline analysis of the packets to and from the HTTP server.

The transmission is done through a network emulation node using the built-in Linux Kernel *netem* module capable of altering the network QoS parameters, such as packet delay distribution, packet loss rate, or transmission throughput. Random packet losses are limited due to access network link layer and transport protocol retransmissions. However, through this mechanisms, loss acts as another source of

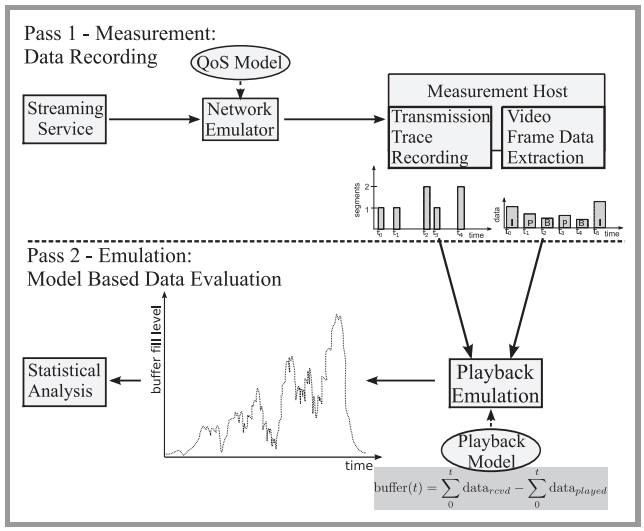


Fig. 3. Two-pass measurement environment used to capture network traces independently of playback strategies.

delay and jitter. We focus on asymmetric access networks, such as ADSL phone lines, which are assumed to form the bottleneck links.

In the second phase, the video file with the frame dataset are re-assembled. Then they are used to feed models in a media playback emulation process, which combines the transmission and video frame traces to calculate the playback buffer fill level for every point in time during the playback. Each frame has its own playing time which specifies the time at which the frame should be played in relation to the initial frame. From this analysis, we can determine how much data is required to play each frame of the video without any delay to allow for an uninterrupted playback. On this basis, we generate statistics about user-perceivable artifacts such as re-buffering events that would occur during the playback. These statistics can then be compared to the results of other models and network QoS.

For our experiment we used a 92 s video file encoded at 850 Kbit/ps.

4.1. Quality Measures

From the user's perspective, the key performance characteristic of a network is the QoS of received multimedia content. However, in the case of HTTP video the transmission is reliable, so there is no packet loss induced video degradation. Nevertheless, packet losses introduce additional delay caused by TCP retransmissions which consequently can lead to re-buffering events resulting in jerky playback. The packet delay and loss reduce also TCP throughput. When the throughput is lower than the playback rate and the buffer has drained, the video playback will pause and wait for new video data. A user expects that delays resulting from content buffering will be minimized and do not occur during normal video play.

Thus, to characterize the relationship between the network QoS and application QoS, for our purpose, we use two

measures for HTTP videos. The first measure of the application QoS takes into account relative total stalling time experienced by a user and is defined as:

$$\mathrm{SR} = \sum_i \Delta t_i / T, \tag{1}$$

where t_i are times for which $N(\Delta t_i)$ has negative value and T denotes a total duration of the video file when played without interruptions. As the above measure is the ratio of total stalling time to the the video duration, it is desirable to minimize its value by an ISP.

The application QoS defined in (1) did not differentiate between the cases in which a user can experience one long stalling period Δt^l or several shorter stalling periods Δt^s where $\Delta t^l = \sum_i \Delta t_i^s$. Thus, in our analysis we also use a second, complementary measure which value is the number of re-buffering events i associated with every stalling period.

In our experiment every video playing scenario has at least one re-buffering events which is a result of an initial buffering. The initial buffering is used to accommodate initial throughput variability or inter-packet jitters. Some streaming strategies may achieve smoother streaming with larger initial buffering, nonetheless it increases the startup latency of received video content. The re-bufferings, which take place in the middle of video playback, are usually a consequence of the congestion avoidance algorithm of the TCP.

In our analysis we compared the SR (1) and stalling frequency for the earlier mentioned buffering algorithms: Flash, HTML5 and simple buffering strategy (Simple). The last strategy assumes that the algorithm always starts playback as soon as any data is available in the buffer. This means that, if the player is currently stalling and a complete frame becomes available in the buffer, playback will immediately restart and the frame will be shown even if this means stopping playback after that frame again. This results in the lowest required buffer space. Moreover, playing the video as soon as possible, gives the fastest end. Consequently, the Simple strategy give the lowest SR and an upper limit for the number of stalls occurring. Conversely, the best way to minimize the number of stalls is to wait for the entire file to be downloaded.

5. Results

5.1. Delays

The delay in an experienced by video content consists of two components: delay introduced by network, which is the time it takes a data packet to travel from sender to receiver and TCP-level delay, which is a consequence how the TCP reacts to fluctuations in the effective network throughput. While throughput fluctuations can occur due to application-level flow control, they are primarily the result of network congestion.

As results of our experiments we obtained statistics of buffer occupancy as a function of time for the three examined playing strategies. An exemplary trace of the buffer occupancy for the Simple strategy is presented in Fig. 4.

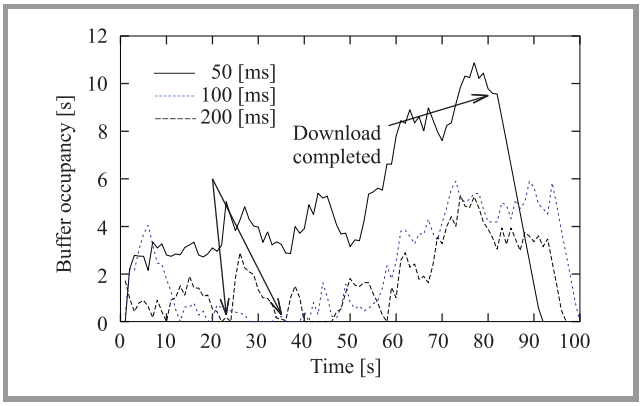


Fig. 4. Player buffer occupancy as a function of time.

We may notice that with increasing latency the buffer occupancy, measured as video playback time, is decreasing and re-buffering events happen more often. When the packet delay is 50 ms, there is only one stalling event on the beginning of the video transmission. According to the formula throughput $\sim 1/\mathrm{delay}$ describing theoretical TCP throughput, in this case the delay is the lowest, thus the theoretical connection throughput is the highest. Therefore, the download of the whole video finishes after about 80 s of the experiment. From this moment the buffer is no longer supplied. It decreases at a constant rate as the video player pulls the remaining video data and presents it to its user.

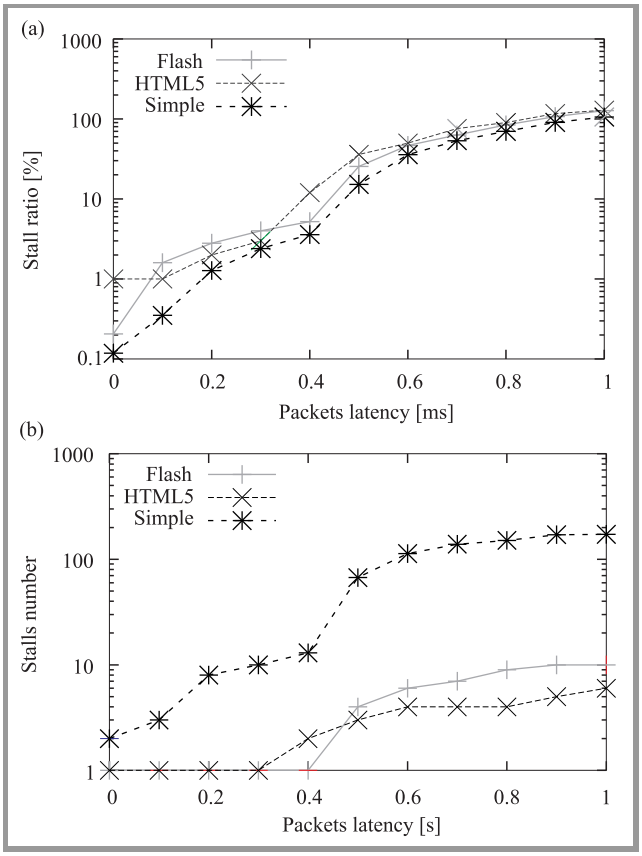


Fig. 5. The influence of packet delay: (a) stalling ratio; (b) re-buffering events.

However, when the delay rose to 100 ms or 200 ms, watching the video is quite inconvenient due to the frequent buffer under-runs. In these situations more re-buffering events occur, the total playing time exceeds the original video length and the file downloading completes after about 92 s.

Generally, we were interested not in a transient buffer analysis but in its examination in the context of application QoS for which measures were defined in the Subsection 4.1. Thus, in the further experiments, except for the packet delay, we obtained statistics of the buffering behavior in scenarios with additional packet loss and network throughput limitation.

As it is shown in Fig. 5(a), packet delay has a certain influence on application QoS which is defined as the SR in Eq. (1). Increasing gradually the packet delay up to 1000 ms caused the successive rise of the SR from less than 1% to about 100% in average. From the three examined playback strategies, the Simple strategy experienced the lowest SR while the highest SR was obtained by Firefox HTML5 strategy. Nonetheless, with increasing delay, the differences between the playing algorithms diminished.

When it comes to measuring the number of stalls, the situation looks quite different. When increasing latency up to 300 ms, a user using the Flash or HTML5 strategies usually experienced only a single re-buffering event which occurred at the beginning of the playback. When the delay exceed 500 ms the HTML5 strategy has the lowest number of stalls from all the three examined strategies. The Simple strategy was not able to successfully mitigate the network impairments which resulted in several re-buffering events during the playback, Fig. 5(b).

5.2. Packet Loss

Packet loss can be caused by a number of factors including signal degradation over the network medium due to multi-path fading, packet drop because of channel congestion, corrupted packets rejected in-transit, faulty networking hardware, faulty network drivers or normal routing routines. When transmitting HTTP video, in the event of packet loss, the receiver asks for retransmission or the sender automatically resends any segments that have not been acknowledged. Nevertheless, retransmitting missing packets causes the throughput of the connection to decrease due to the sliding window mechanism used for acknowledgement of received packets, implemented in the TCP.

For the packet loss up to 2% the SR graph resembles *S* shape, Fig. 6(a). For packet loss below 0.8% the SR has value 1 for the Flash and HTML5 strategies, and about 0.2 for the Simple strategy. Such values of the SR can be considered relatively low and should not have much impact on the received video quality. However, for the packet loss between 1% and 1.2% the SR rises rapidly achieving values of several tens. The further increase of the packet loss rate results in a relatively small rise of the SR. Generally, the Simple strategy has the lowest value of the SR. We can also notice that for 1% packet loss there is a significant difference between the Flash and HTML5 strategies which diminishes for the other packet loss values.

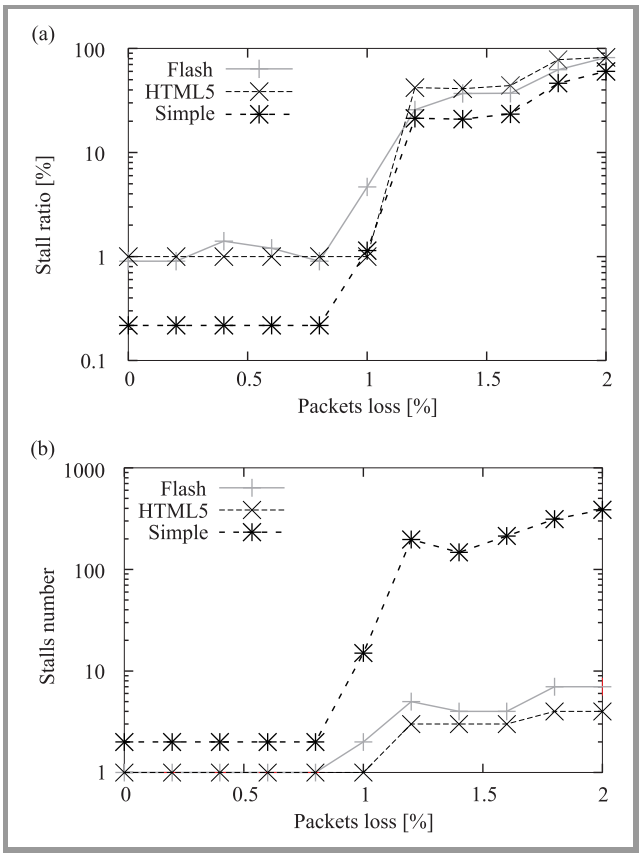


Fig. 6. The influence of packet loss: (a) stalling ratio; (b) re-buffering events.

When to measure the video streaming resilience against the packet loss as number of stalls, Fig. 6(b), the shape of the chart is similar to the shape of the SR presented in the Fig. 6(a). We can also observe here the steady rise of the stalls number when the packet loss is lower than 0.8% and higher than 1.2%. For the packet loss value between 0.8% and 1.2% the stalls number grow quite fast. Contrary to the results presented in the Fig. 6(b), this time the Simple strategy has the worst performance. The HTML5 strategy is little better than the Flash strategy for the packet loss higher than 1%.

5.3. Throughput

In this section we investigate how the download throughput limitation influence the YouTube video streaming. The upload throughput in the experiments was fixed and set to 10 Mbit/s. Figure 7(a) shows the dependency between the SR and download throughput ranging from 256 Kbit/s up to 10 Mbit/s. We can observe that the 1 Mbit/s download throughput is sufficient for our streamed video independently of the playing strategy used. Increasing the throughput beyond 1 Mbit/s does not significantly improve the SR. From the other side, even a small throttle of the network throughput results in a dramatic rise of the

SR value. Taking into account that the video encoding rate is 850 Kbit/s, such streaming behavior is common sense and the network throughput below this threshold value should be insufficient. Furthermore, the mentioned threshold will be additionally increased by network protocols overhead.

The similar situation is when we used the stalls number measure, Fig. 7(b). For the 1 Mbit/s network throughput and and above the number of stalls is 1 for the Flash and HTML5 strategies. The Simple playing strategy experiences two re-buffering events.

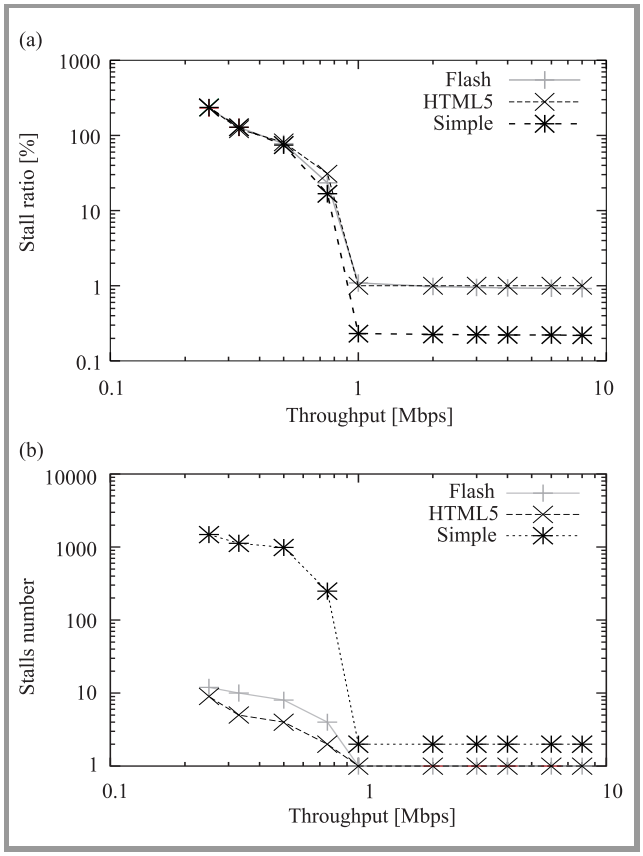


Fig. 7. The influence of throughput limitation: (a) stalling ratio; (b) re-buffering events.

Generally, we can conclude that the all three playing strategies cannot satisfactorily cope with even small limitation in the network throughput, and the initial buffering implemented by the Flash and HTML5 strategies fails in this situation.

6. Conclusions

In the paper we tried to answer how network defects, manifested as latency, packet loss and throughput limitations impacts the quality of HTTP based video playback. For this purpose, we conducted an experimental evaluation of YouTube video transmission examining the quality of experience of end user applications expressed as a function of playback buffer occupancy. Through this analysis we investigated how long the video playback is stalled and how often re-buffering events take place.

Generally, in order to watch the 850 Kbit/s video without interruptions and extensive buffering time, the packet delay introduced by the network should not exceed 200 ms. The packet loss higher than the 0.8% makes the viewing of online video very inconvenient. For smooth transmission of the video, network connection throughput should be at least 1 Mbit/s.

Our analysis revealed that there exists some small differences between the Flash and HTML5 strategies, however, in the most cases they will remain unnoticed by the end user. The buffering algorithm used in an HTML5 player showed the highest resilience against the packet delay and loss when taking into account the number of re-buffering events experienced during the video play. However, when comparing these both strategies with the Simple strategy, it is obvious that starting the video playback as soon as a minimum buffer level is achieved is insufficient. Although the Simple strategy has lower stalling time compared to the other two strategies, nonetheless, the number of re-buffering events which occur during the video streaming in unacceptable for an end user. All the three playing strategies cannot satisfactorily cope with even small limitation in the network throughput. The initial buffering mechanism implemented by the Flash and HTML5 strategies fails in that situation.

Acknowledgment

The research was partially supported by the National Science Centre (Poland) under grant DEC-2011/01/D/ST6/06995.

References

[1] "Global mobile data traffic forecast update, 2010002015", *Cisco Visual Networking Index. White Paper*, Cisco, 2011.

[2] A. Rao, Y. S. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous, "Network characteristics of video streaming traffic", in *Proc. 7th Int. Conf. Emerg. Netw. Exper. Technol. CoNEXT 2011*, Tokyo, Japan, 2011.

[3] S. Alcock and R. Nelson, "Application flow control in YouTube video streams". *ACM SIGCOMM Comp. Commun. Rev.*, vol. 41, no. 2, pp. 24–30, 2011.

[4] I. Hickson, "HTML5: a vocabulary and associated APIs for HTML and XHTML', April 2010 [Online]. Available: http://www. w3.org/TR/html5/

[5] A. Finamore, M. Mellia, M. Munafo, R. Torres, and S. R. Rao, "YouTube everywhere: impact of device and infrastructure synergies on user experience", Tech. Rep. 418, Purdue University, May 2011.

[6] S. Acharya and B. C Smith, "Experiment to characterize videos stored on the web", in *Proc. ACM/SPIE Multimedia Comput. Netw. MMCN 1997*, vol. 3310, pp. 166–178, 1997.

[7] S. Acharya, B. Smith, and P. Parns,"Characterizing user access to video on the world wide web", in *Proc. ACM/SPIE Multimedia Comput. Netw. MMCN 2000*, vol. 3969, pp. 130–141, 2000.

[8] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet", SIGCOMM Comput. Commun. Rev., vol. 29, no. 4, pp. 189–200, 1999.

[9] S. Floyd, M. Handley, J. Padhye, and JÃ¶rg Widmer, "Equation-based congestion control for unicast applications", *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 43–56, 2000.

[10] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge". in *Proc. 7th ACM SIGCOMM Conf. Internet Measur. IMC 2007*, San Diego, CA, USA, 2007, pp. 15–28.

[11] M. Zink, K. Suh, Y. Gu, and J. Kurose, "Characteristics of YouTube network traffic at a campus network-measurements, models, and implications", *Computer Netw.*, vol. 53, no. 4, pp. 501–514, 2009.

[12] L. Plissonneau, T. En-Najjary, and G. Urvoy-Keller, "Revisiting web traffic from a DSL provider perspective: the case of YouTube", in *Proc. ITC Spec. Seminar Netw. Usage Traffic*, Berlin, Germany, 2008.

[13] M. Cha, H. Kwak, P. Rodriguez, Y. Y Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system", in *Proc. 7th ACM SIGCOMM Conf. Internet Measur. IMC 2007*, San Diego, CA, USA, 2007, pp. 1–14, 2007.

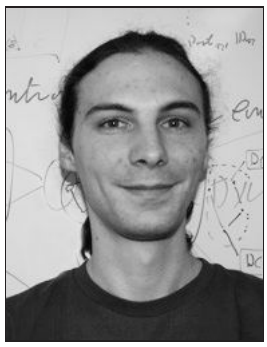
[14] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of YouTube videos", in *Proc. 16th Int. Worksh. Quality of Service IWQoS 2008*, Enschede, The Netherlands, 2008, pp. 229–238.

[15] A. Abhari and M. Soraya, "Workload generation for YouTube", *Multimedia Tools and Appl.*, vol. 46, no. 1, pp. 91–118, 2010.

[16] D. K. Krishnappa, S. Khemmarat, and M. Zink, "Planet YouTube: global, measurement-based performance analysis of viewer;'s experience watching user generated videos", in *Proc. IEEE 36th Conf. Local Comp. Netw. LCN 2011*, Bonn, Germany, 2011, pp. 948–956.

[17] S. Benno, J. O. Esteban, and I. Rimac, "Adaptive streaming: the network HAS to help", *Bell Labs Tech. J.*, vol. 16, no. 2, pp. 101–114, 2011.

Arkadiusz Biernacki received the M.Sc. and Ph.D. degree in Computer Science from the Silesian University of Technology, Poland, in 2002 and 2007, respectively. From 2007 he is an Assistant Professor at the Silesian University of Technology. From 2010 he has been collaborating with Chair of "Future Communication" (endowed by Telekom Austria) at the University of Vienna. His research interests focus on network traffic modeling and computer system simulations.
E-mail: arkadiusz.biernacki@polsl.pl
Institute of Computer Science
Silesian University of Technology
Akademicka 16
44-100 Gliwice, Poland



Florian Metzger is a research assistant at the the Chair of "Future Communication" (endowed by Telekom Austria) at the University of Vienna. He received his diploma thesis at the University of Wuerzburg, Germany, in 2009. His research interests cover signaling load in mobile networks as well as modern video streaming approaches.
E-mail: florian.metzger@univie.ac.at
Chair of Future Communication
University of Vienna
Währinger Str. 29/5.46
1090 Vienna, Austria

Kurt Tutschku holds the Chair of "Future Communication" (endowed by Telekom Austria) at the University of Vienna. Before that, he was an Assistant Professor at the Department of Distributed Systems, University of Wuerzburg. He led the department's group on Future Network Architectures and Network Management until December 2007. From February 2008 to July 2008, he worked as an Expert Researcher at the NICT (National Institute for Information and Communication Technology, Japan). He has received a doctoral degree in Computer Science from University of Wuerzburg in 1999 and completed his habilitation at the University of Wuerzburg in 2008. His main research interest include future generation communication networks, Quality-of-Experience, and the modeling and performance evaluation of future network control mechanisms and P2P overlay networks.
E-mail: kurt.tuschku@univie.ac.at
Chair of Future Communication
University of Vienna
Währinger Str. 29/5.38
1090 Vienna, Austria